ASHESI UNIVERSITY COLLEGE

THE MOLAM PROJECT: THE DESIGN OF A CONTEXT SPECIFIC NAVIGATION FRAMEWORK FOR A MOBILE ROBOT

DANIEL NII TETTEY BOTCHWAY

Dissertation submitted to the Department of Computer Science,

Ashesi University College

In partial fulfillment of Science degree in Computer Science

APRIL 2013

Declaration

I hereby declare that this dissertation is the result of my own original work and that no part of it has been presented for another degree in this university or elsewhere.

Candidate's Signature: ……

Candidate's Name : ……

Date : ……

I hereby declare that the preparation and presentation of the dissertation were supervised in accordance with the guidelines on supervision of dissertation laid down by Ashesi University College.

Supervisor's Signature: ……

Supervisor's Name : ……

Date : ……

Acknowledgements

This dissertation would not have been possible without the guidance and the help of several individuals who in one way or another contributed and extended their valuable assistance in the preparation and completion of this project.

First and foremost, my utmost gratitude goes to my supervisor and first robotics lecturer, Dr. G. Ayorkor Korsah whose insightful directions and assistance made this project a rewarding journey.

I am truly grateful to Mr. Aelaf Dafla, Dr. Nathan Amanquah, Prof. Edwin Kay, and basically all the lecturers in the Computer Science Department for their immense contributions to shaping my mind.

I would also like to thank my colleagues and friends in the Computer Science Class of 2013 for all their inspiration as we overcame all obstacles during the last four years.

Last but not the least; I would like to thank my family and the one above all of us, the omnipresent God. Thank you so much, dear Lord, for answering my prayers and giving me the strength to plod on despite my constitution wanting to give up and throw in the towel.

Abstract

Ashesi University College recently acquired a TurtleBot[1] robotic platform to experiment with service robotics on campus. This platform runs the Robotics Operating System (ROS)[2] for developing robotics applications. This MOLAM project (Motion, Localization And Mapping) sought to build a fundamental framework that extends ROS functionalities on a TurtleBot to navigate autonomously around the Ashesi campus in Berekuso.

The focus of this project was to extend the ROS enabled functionalities on the TurtleBot for mapping, localization and motion planning to create a customized navigation framework suited for the Ashesi University campus. Having such a foundation, other service applications, such as a waiter robotic system, a courier robotic system, a tour guide robotic system etc., can be built on this framework.

This paper outlines the various tools and processes used in assembling and configuring this robotic system. It also delineates the challenges encountered in assembling and configuring this robotics system, as well as the "work-arounds" devised to solve these challenges. Finally, this paper proposes design concepts for future work in using the TurtleBot system as a Tour Guide Robotic System for Ashesi University College.

[1] http://turtlebot.com/
[2] http://ros.org/

Contents

List of Figures

List of Abbreviations

Abbreviation	Meaning
API	Application Programming Interface
DDR3	Double Data Rate 3
DIN	Deutsches Institut für Normung
GB	Gigabyte
HRI	Human Robotics Interaction
IOS	International Organization for Standards
IP	Internet Protocol
KDL	Kinematics And Dynamics Library
LIDAR	Laser Detection And Ranging
MOLAM	Motion Localization And Mapping
OS	Operating System
PCL	Point Cloud Library
PGM	Portable Gray Map
QR	Quick Response
RFID	Radio Frequency Identification
RGB	Red Green Blue
RGBD	Red Green Blue Distance
ROS	Robotics Operating System
SLAM	Simultaneous Localization And Mapping
USB	Universal Serial Bus
WIFI	Wireless Fidelity
WLAN	Wireless Local Area Network

Chapter 1: Introduction

1.1 Introduction

The MOLAM project is a robotics research project that is aimed at implementing a navigation framework for a TurtleBot (a research robot) that was acquired for this project. This framework will be built on ROS, a meta-operating system that "provides libraries and tools to help software developers create robot applications" (Foote, 2013). The first application of this framework will be to integrate it into a robotic tour guide system for Ashesi University College's new campus at Berekuso. This robotic tour guide would relieve secretaries of the extra role they play in giving visitors a tour of the new campus. It would also give visitors a unique robotics experience.

This project is therefore focused on configuring the TurtleBot and extending its navigation functionalities to suit the context of an Ashesi environment. The framework to be developed will result in a very robust robotic infrastructure that is capable of accurate mapping and path planning using higher level information such as the actual names of offices and building structures on the campus. Moreover, this infrastructure should be an independent module with easy integrability into future robotics projects that would require a locomotion and navigation system. The main focus areas of the project which will forge the core underlying functionalities of this framework would be MAPPING, LOCALIZATION and MOTION PLANNING; which are also the inspiration behind the name of the project MOLAM *(**MO-**tion panning, **L-**ocalization **A-**nd **M-**apping).*

It must be noted that a similar project was undertaken in 2012 by a student; Kwame Afram to build a tour guide system for Ashesi using the Lego NXT robot kit (Afram, 2012). However, due to the size, sensor limitation and limited navigation capabilities of the Lego NXT robot, the TurtleBot is more suited for campus-scale navigation.

1.2 Objectives

This section describes the goals of this project in developing the context-specific navigation framework. These goals were considered by relating the functionalities of the proposed navigation framework to the existing capabilities of the ROS-TurtleBot platform.

1.2.1 Setting Up ROS-TurtleBot

The robotics technologies (i.e. both the TurtleBot and the ROS platform) obtained for this project are entirely new to the university. The first focus of this project will be to configure these systems to a working level that will allow robotics application to be developed on it. A major part of this project will be to ensure that the ROS-TurtleBot is correctly setup for future robotics development.

1.2.2 Mapping

The fundamental concept of robotic mapping is that for a given environment, a robot should be able to use the available on board sensors it has to gather data about its environment and translate that data into a map model. The map model should accurately represent the environment with the necessary information to identify the various objects present. ROS has a mapping tool that is capable of a generating a map model of an

environment that represents obstacles and free space around the robot. This framework seeks to extend this map model by overlaying it with actual structural information of the campus to give users a human level interpretation of the map. With the university's plans to extend the campus, the robot should be able to build a new map from scratch, modify an existing map to accommodate changes in the environment, and repair parts of the map if broken.

1.2.3 Localization

Localization is a concept that focuses on the ability of a robot to determine its current position in a given map, based on the information it has from its environment and the map. Localization can prove to be challenge in the sense that, if the robot has an ever changing map or environment, knowing exactly where it is at any point in time could be difficult. The concept of Simultaneous Localization and Mapping (SLAM) solves this challenge using various techniques. SLAM is the process of building a map of an unknown environment while simultaneously determining the location of the robot in the map. The localization challenge can also be addressed by having artificial landmarks help the robot determine where it is by embedding information in data technologies like RFID tags and QR-Codes in the environment (Rusu, Gerkey, & Beetz, 2008). This project will focus on using a SLAM GMapping[3] technique implemented by ROS in its localization mechanism. Pinpointing the location of the robot will be done on two levels. The first level will be using the location coordinates as represented internally by the robot's map model and the second level will

[3] http://www.openslam.org/gmapping.html

be a high-level representation for a user to understand. For instance the coordinate point (123,432) may be interpreted and represented as a point in the Ashesi Library. A translation interface will therefore be needed to convert between the two locations representations.

1.2.4 Motion Planning

Given a map and the current position of the robot; the robot should be able to generate a sequence of points (a path) in the map that will lead it to a desired destination point. The path should be feasible for the robot to traverse (if some parts of the environments have staircases the robot should avoid them) and optimal in terms of determining the shortest (or most cost effective) path to use. Consequently, the motion planning mechanism of this framework should allow a robot adequately acknowledge the presence of other objects in the robots environment and respond appropriately to them. That is, the robot should not collide with moving objects in its environment and if any object blocks the path of the robot, a new alternative path should be generated to get the robot to the destination location. Also, since the proposed navigation framework will be providing high-level information map, path planning will occur at both high-level locations on the information map and low-level coordinates on the robot's internal map models.

1.2.5 Developer API

The project will provide technical documentation so that developers interested in building upon this framework, can easily interface it with their work. This documentation or Application Programming Interface (API) will be a software application interface of the navigational and

locomotion functionalities this framework provides, that can be used in building higher level robotics applications. The API will conceal the complexities entailed in implementing the functionalities and provide only the necessary interface to allow seamless integration.

1.2.6 Visualization Tool

Apart from the API, this project will also develop a friendly user interface to allow easy interaction with the framework itself. The data and modeling done internally in the robots are largely represented by several numeric values that seem to make no sense. The interface to be created will be more of a data visualization tool that will provide a graphical representation of the internal data processes. With this tool, higher level interpretation and analysis can be done concerning the functioning of the framework.

1.2.7 Additional Documentation

This project will add to the knowledge base and the current reference materials in the robotics field especially in the areas concerning navigation using SLAM techniques. The project will make available the necessary documentation on research procedures, solution concepts, challenges, "work-round's" and results of experimentations carried out throughout the project to serve as guiding principles to any other interested researcher.

1.3 Scope

As outlined in the Objectives section above, the goal of this project is not to build a robot from scratch. Rather the aim is to assemble and configure the TurtleBot to use the ROS framework. Having done that, the project

will also utilize the functionalities provided by the ROS framework to create a non technical user friendly interface which obscures the technical details of the framework. Nonetheless, this interface should extend the ROS functionality to be well suited for the Ashesi University College campus in terms of the representation and interpretation of the various map data models. This project will not concern itself with the Human Robotic Interaction needed for a successful robotic tour guide system, but will provide the interface and tools as permitted by the ROS framework to develop application systems not limited to only a tour guide system.

1.4 Motivation

This project is inspired by the author's interest in current trends in the robotics industry which is shifting the traditional understanding and implementation of robots to a more complex frontier. The first generational understanding and usage of robots identified robots as automated machines installed in well-defined and controlled environments like factories where they had little or no interaction with dynamic objects (like humans or other robots) in the environment. Currently, the field of robotics is moving these machines from their own world into a more complicated world, where it is necessary to appreciate and respond appropriately to the interactive and dynamic nature of the environment and the objects within it. In this liberating yet complex environment, robotics research is pushing the limits to the kind and quality of assistance a machine could offer to the human society. Having said that, the great capabilities of new generational robots rely on an essential piece, which is their ability to move around in a sufficiently autonomous, safe and

intelligent manner. This exciting yet computationally intense design required to have such an intelligent system, stirred up passions to embark on this project with the aim of having a sense of accomplishment and satisfaction while making a little contribution to the robotics society.

Chapter 2: Background

This chapter describes the technology used in this project and summarizes other related works. The robotics projects looked at in this chapter were related to navigation using the Kinect sensor kit and to public area service robots specifically, a tour guide museum robot.

2.1 Technology

2.1.1 TurtleBot

TurtleBot[4] is a robot kit product from Willow Garage[5] that is built from various hardware components for cheap personal robotic applications. This robotic kit offers a simple and easy-to-use system suited for robotic research projects. The TurtleBot is made up from off-the-shelve electronic hardware pieces which include:

- iRobot Create
- Mounting structure
- Microsoft Kinect kit
- Sensor board
- Netbook computer

2.1.1.1 iRobot Create

iRobot Create is a robotic hardware platform manufactured by iRobot Corporation. This platform is modeled after the Roomba domestic vacuum

[4] http://turtlebot.com/
[5] http://www.willowgarage.com/

cleaning robot (Figure 2.1). The Create platform does not come with the vacuum cleaner hardware but rather an empty space to serve as a cargo bar with a serial pin port for digital and analog input and output devices. The Create was specifically designed to support various robotic hardware accessories needed in robotics research developments.

The Create is a differential drive robot (that is a robot with two powered wheels) with a third point of contact and an optional fourth wheel for stability. The Create base has an Omni-directional Infrared receiver, a Mini-DIN port, a cargo bar for robot hardware accessories and a charging port. The Create also has cliff sensors to detect if the robot wheels are in contact with the ground or not and bumper sensors to detect collision with solid objects. The Create base also comes with a Mini-DIN connector cable to connect the hardware platform to a computer.

Figure 2.1 - iRobot Create Base

2.1.1.2 Mounting Structure

The TurtleBot has a set of laser cut plates and set of custom made standoff kit frames (Figure 2.2). These are fitted together to form a shelve-like structure to hold the Kinect kit, the Netbook computer and other hardware accessories.

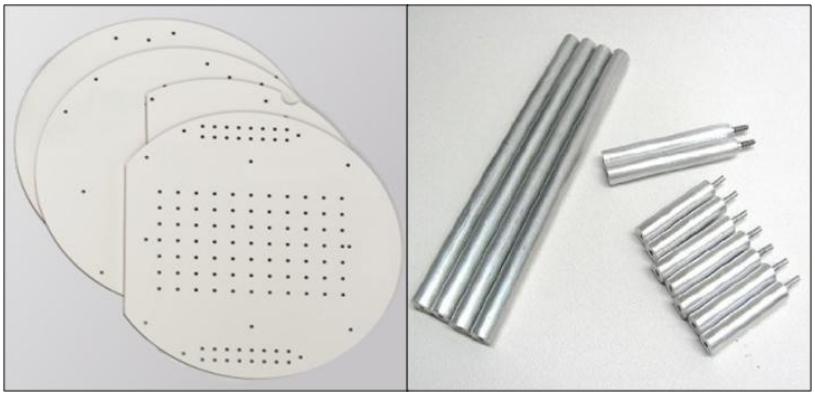

Figure 2.2 - TurtleBot Mounting Structures

2.1.1.3 Microsoft Kinect Kit

Microsoft in 2010 released a 3D motion sensor input device for the Microsoft Xbox 360 game console; the Kinect Kit (Figure 2.3). This kit has

- A RGB color camera
- An Infrared laser projector
- An Infrared laser sensor
- An array of microphones and
- A motorized pivot base.

This kit works by using an infrared projector to throw an infrared laser pattern on the objects in its view and the infrared laser "sensor picks up the laser to determine the distance of each pixel" (Ackerman, 2011). Overlaying the depth sensor values of the images from the RGB camera gives an RGBD (RGB-Distance) image that can "map out body gestures, positions, motions and generate 3D maps" (Ackerman, 2011).

The Xbox peripheral has not only been embraced by the gaming community but by many robotics hobbyists, who have turned the device into a desirable robotics sensor kit. With the relatively cheap price and

great sensing capabilities, the Kinect has become the primary sensor kit for many TurtleBot robotics projects.

Figure 2.3 – Microsoft Kinect Kit

2.1.1.4 Sensor Board

The TurtleBot's main external electronic board is a power and sensor circuit board with a Gyro (Figure 2.4). This board fits into the Create base's serial pin port to supply power to the Kinect and sensor information from the Gyro to a computer. The board provides a regulated 12V supply to power the Kinect Kit and a single axis gyro that can measure yaw rate up to 250 degrees/second.

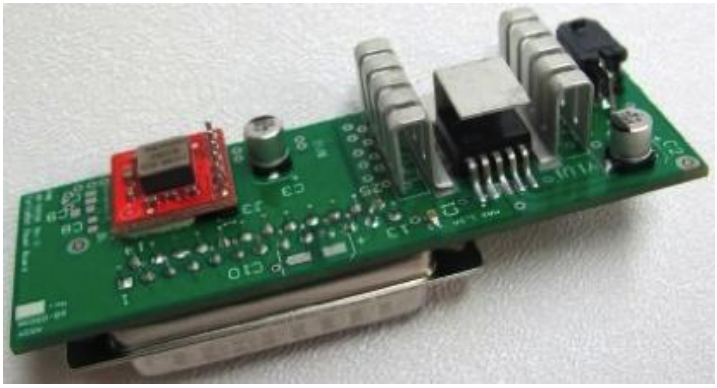

Figure 2.4 – TurtleBot Sensor/Power Board

2.1.1.5 Netbook Computer

The TurtleBot uses a computer to process most of the computational requirements of the robotic system. Though the Create base has a

processor, a computer provides higher and faster computational capabilities required in handling the various data sets from the sensors. The computers used by the TurtleBot are Netbook computer because they have a desirably small size. However, any laptop computer that is small enough to fit into the bottom shelf of the TurtleBot shelf structure can be used. For instance a Lenovo T240 ThinkPad laptop was used for this project. The minimum hardware specifications of the needed computer are listed as follows:

- 2GB DDR3 memory
- Dual core processor (at least 1.8GHz)
- WLAN 802.11b/g/n (@ 2.4GHz)
- 2 USB port (with at least a USB port 2.0 for the Kinect)
- 250GB storage

2.1.2 Robotic Operating System (ROS)

The complexity of writing software for robots has led to the development of several robotics software frameworks of which ROS is an example. ROS implements the fundamental operations of an Operating System (OS) by managing hardware resources of a robotics system but it does not handle the scheduling and management of processes. Since ROS is installed and run on a host OS, the host OS handles the process management requirements. ROS was developed by Willow Garage to achieve certain design goals in a robotics framework which include supporting a peer-to-peer network structure, allowing development using multi programming languages, being thin (i.e. having standalone libraries that are independent of ROS), having several tools to accomplish tasks, and being open source (Quigley, et al., 2009).

ROS connects robot hardware to software to create an advance programming environment by providing services for hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management (Ruhland, 2012). With a framework like ROS, robotics researchers only need to focus on higher-level manipulation of robots and thus are able build complex systems faster.

2.2 Related Work

2.2.1 NXT-Kinect-SLAM

Recent work that is similar to the MOLAM project is an implementation of a navigation system for an indoor mobile robot using the NXT robotics kit and a Kinect. The paper *An Investigation of the Use of Kinect Sensor for Indoor Navigation* (Keat & Ming, 2012), describes the hardware and software architecture of the robot and discusses the results of using a low cost robotic setup. In the setup, a four-wheel robot was built using the NXT robotics kit with a Microsoft Xbox 360 Kinect sensor mounted on the robot (Figure 2.5). The depth images from the Kinect sensor are converted to a 2D laser scan output using a laptop running ROS. The laser scan and odometry readings from the robots base were used as input to build a map using SLAM GMapping technique[6]. The navigation stack plans and sends commands to autonomously navigate the robot after generating a map. For better performance, the robot uses A* algorithm (Fradj, 2011) to generate a high level global path in the global planner and the path is fine-tuned using a local planner that uses Dynamic Window Approach for collision avoidance (Foxy, Burgardy, & Thrun, 1997).

The researchers found out that the use of the NXT-Kinect setup was not affected by different lighting situations, which meant the robot worked well even in dark environments. Compared to expensive sensor equipment like LIDAR used by Google's autonomous car and Hokuyo Laser sensor, the Kinect which is cheaper, has similar working range and similar 3D

[6] http://www.openslam.org/gmapping.html

scanning capabilities. Moreover, optimizing the position of the Kinect sensor increased the accuracy of the SLAM mapping to about 99.6%, thus making the Kinect suitable for autonomous robot navigation (Keat & Ming, 2012).

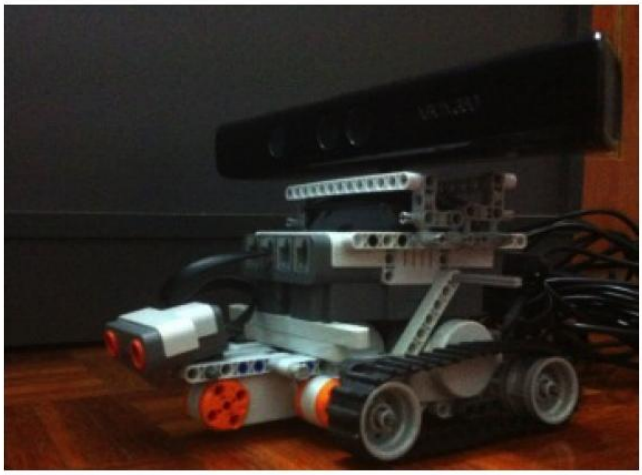

Figure 2.5 – NXT-Kinect Setup

2.2.2 Minerva

There are older robotics research projects that have led to the deployment of exhibition robots and also permanent installation of robots in public spaces. Such robots include tour guide museum robots Rhino (Burgard, et al., 1999), Minerva (Thrun, et al., 1999), and Jinny (Kim, et al., 2004). Minerva, as described in the paper *MINERVA: A Second-Generation Museum Tour-Guide Robot* (Thrun, et al., 1999), is a robot developed by a robotics research team from Carnegie Mellon University that worked on its predecessor; Rhino. The robot was exhibited in the summer of 1998 for two-week at the Smithsonian's National Museum of American History.

Minerva (Figure 2.6) builds two maps for better performance; an occupancy map by probabilistically determining consistent readings of

odometry values and a texture map by using of an upward facing camera at the museum's ceiling to build a large-scale mosaic of the ceiling's texture. For the highly dynamic environment of the museum, Minerva uses an improved version of Markov localization (Dieter Fox, 1999) and Dynamic Window Approach (Foxy, Burgardy, & Thrun, 1997) collision avoidance method for localization and obstacle avoidance respectively. It also uses Coastal Navigation algorithm for path planning. Coastal Navigation is a path planning technique that minimizes path lengths but maximizes the content of information a robot will receive at different points along the path so that it does not get lost (Roy, Burgard, Fox, & Thrun, 1998). The software system used by Minerva was developed to address navigation in dynamic and unmodified environments, allow short-term Human Robotic Interaction (HRI) with visitors and have a virtual tele-presence with tele-operation capabilities over the internet.

Figure 2.6 - Minerva

2.2.3 Relevance of the Related Work

The above mentioned robotics projects are similar to the MOLAM project on certain levels. The MOLOAM project will be using the Kinect and the experimentation by Keat and Ming proves that, the Kinect is a capable 3D sensor kit. The Kinect which is the primary visual senor for the TurtleBot will therefore generate reliable maps for the proposed navigational framework. Also, using the overall design of the robotics system for public space as exemplified in the Minerva project gives great insight on how the proposed navigational framework can be developed. These design outlines serve as guidelines to ensure that the proposed framework is well suited for public space and human interactions.

Chapter 3: System Design

This section describes the systems architectural design of the proposed navigation framework. The fundamental design of the framework will be to extend the capabilities of the ROS platform to support the high-level functionality the framework will provide. The diverse capabilities provided by the ROS platform makes it imperative to identify the specific modules that this framework will interact with.

3.1 System Design Concept

The proposed system for the MOLAM project has a layered architectural design. The components of this system include the Environment, TurtleBot hardware (Kinect and the Create base), ROS framework, the proposed context-specific navigation module, Client programs and the user interface. As shown in Figure 3.1, the blocks in orange color form the core of the proposed navigation framework. The Environment component contains the wide set of objects that the robot would interact with. The Kinect Kit uses its motion and visual sensing capabilities to scan and create a 3D image (Point Cloud data set) of the objects captured in the environment. The ROS implemented PCL (Point Cloud Library) package will then convert the 3D image to a 2D image (laser scan data set). Values from the odometer and the laser scan will then be passed on for the GMapping functionality to create a SLAM map. GMapping is one of the different techniques that are used in solving a SLAM problem and this is already implemented in ROS. The Map Handler and Storage module will process the raw SLAM map into a high-level information map for

visualization on the user's display and for the use by client program such as the tour guide functionality. The robot can be tele-operated using the user's input device (the keyboard) from the workstation to move the robot during the mapping phase.

After the mapping phase, client programs can then request for a path to be planned; for instance, a path from the admissions office to the cafeteria. This command will be received and executed by the Localization and Navigation block. The Localization and Navigation block pulls maps from the Map handlers and Storage module and runs a translation routine to convert the high-level destination locations into appropriate SLAM coordinates. Afterwards, a global path between building structures, and local paths within a single building structure will be planned. These paths are then extracted for navigating the robot to the destination autonomously. During the autonomous navigation phase, the Emergency Control block uses 2D laser scan images and data from the bumper and cliff sensors for obstacle detection and avoidance. When there are no obstacles in a safe distance range the path points are interpreted as movement command for the actuator drivers provided by ROS. These drivers in turn send the appropriate velocity and power values to the motors of the Create base to move the robot.

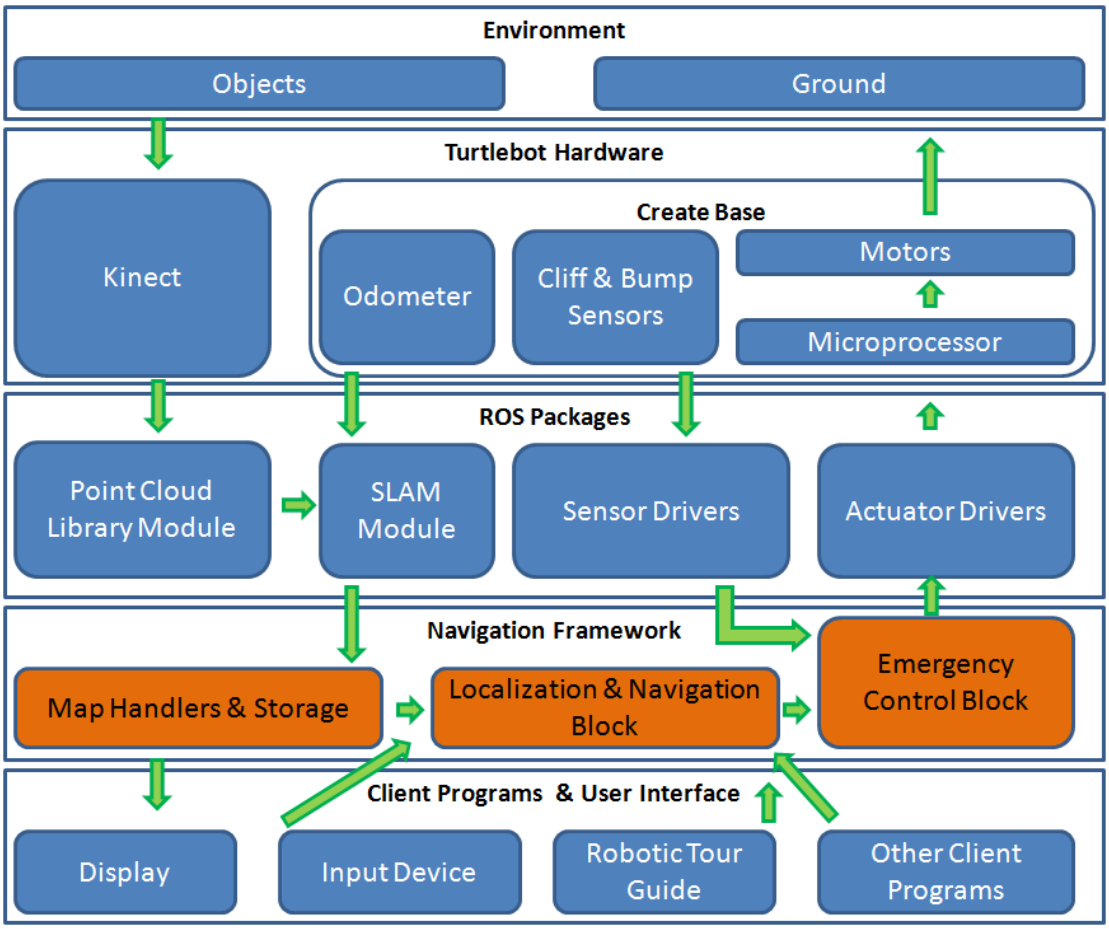

Figure 3.1 – Systems Architectural Design

3.2 Navigation Framework

This section gives a detailed description of the core blocks that make up the navigation framework component of the entire robotic system. This component is made of three blocks; the Map Handlers and Storage, the Localization, and Navigation and the Emergency Control and a proposed design for each is given below.

3.2.1 Map Handlers and Storage

This module handles the SLAM maps created after the mapping stage. The SLAM map (refer to Figure 3.2) generated by the GMapping tool is a PGM (Portable Gray Map) image produced from an occupancy grid map. An occupancy grid map is a matrix that has numeric values indicating the probability of the absence or presence of an obstacle in a given cell. Based on a set probability threshold value, the various cells of the map containing the probabilistic values are labeled as either empty or occupied. This labeled grid map can then be interpreted as an image with different color allocation for empty or occupied cells. The Map Handler and Storage module will process the SLAM maps into an informative map representation that a user can understand. Also, client programs like the tour guide program can use the information map for high-level path planning.

- Firstly, when a SLAM map is passed to this module it will be appropriately renamed and permanently stored.
- This map will then be presented to an operator in a simple third party image editing software. The user will then use the editing tools to

augment the map by manually drawing the diagrams that represent actual building structure in the map. Alternatively, the operator can simply select and label regions of the SLAM map without worrying about the detailed architectural design of the campus.

- The operator will have to tag the various building structures with necessary information that will correspond to pixel locations on the augmented map. The map as shown in to Figure 3.3 becomes an information layer over the actual SLAM map. Moreover, this becomes a user friendly information map that the user can understand and appreciate.
- The global map of the campus is then built using various local SLAM maps of individual building structures (refer to Figure 3.4). This global map will also be captured in the information overlay of the campus.
- The interconnection of the SLAM map in the global map creates a high-level planning graph. As illustrated in Figure 3.5, the blue colored oval blocks represent nodes in the graph. These nodes are SLAM maps of the internal space of a given building structure on the campus. The orange colored round blocks are also nodes, but they represent point locations in the global map at which paths between structures intersect.
- A complete SLAM graph map model as shown in Figure 3.6 will be the global map representation of the university's campus as used and stored internally by the robot.

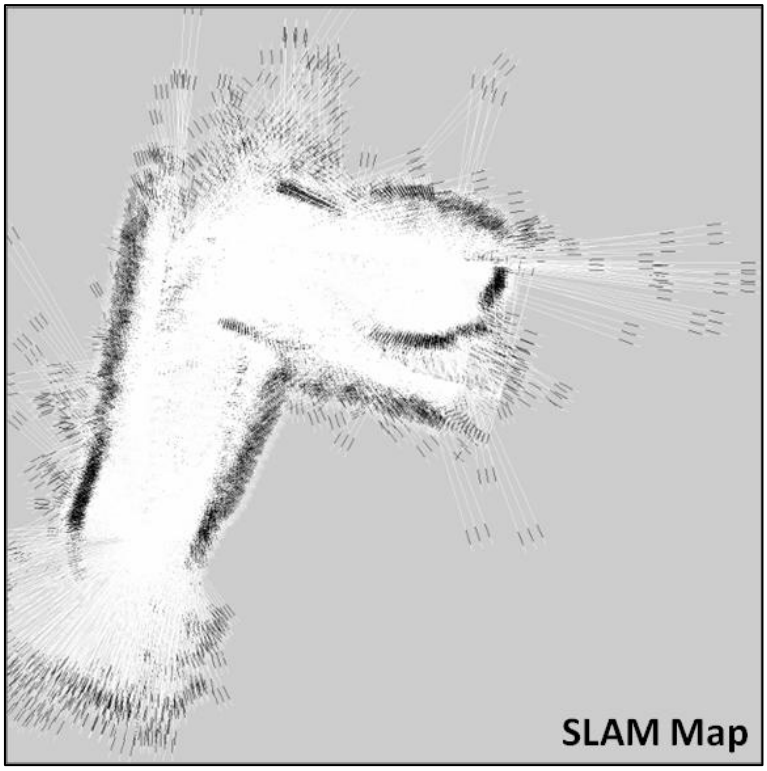

Figure 3.2 - Example SLAM Map

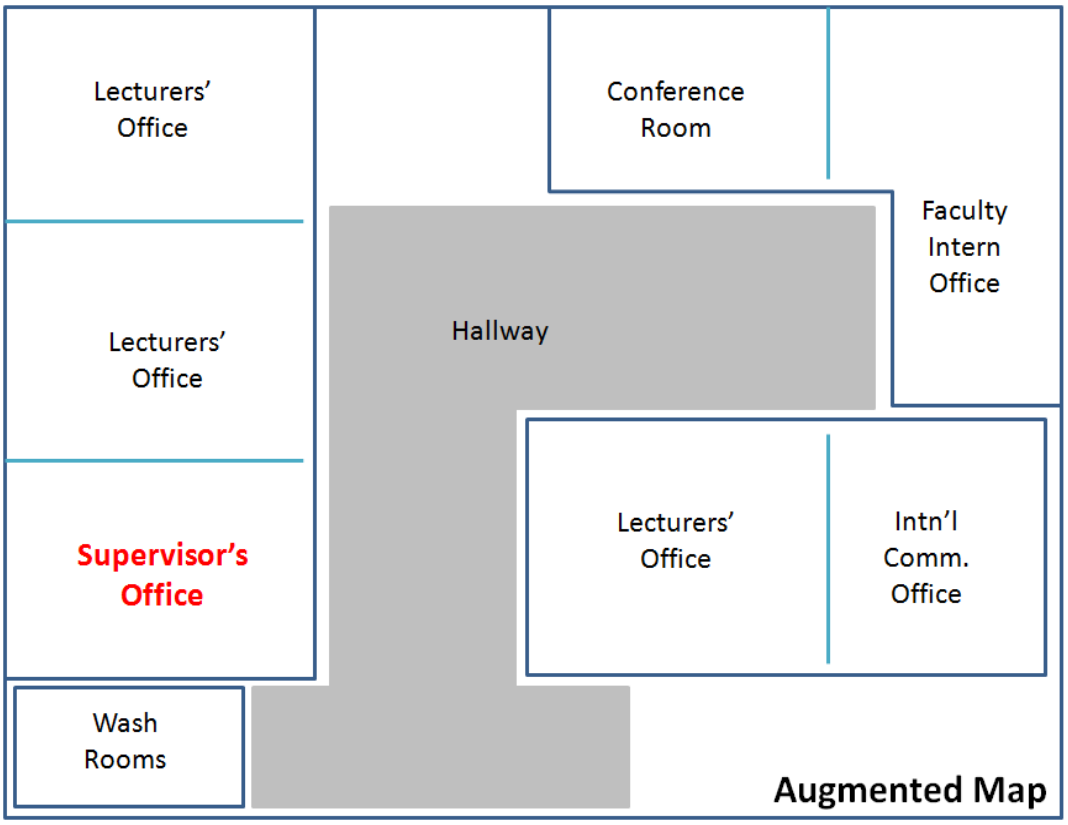

Figure 3.3 – SLAM Map with Information Overlay

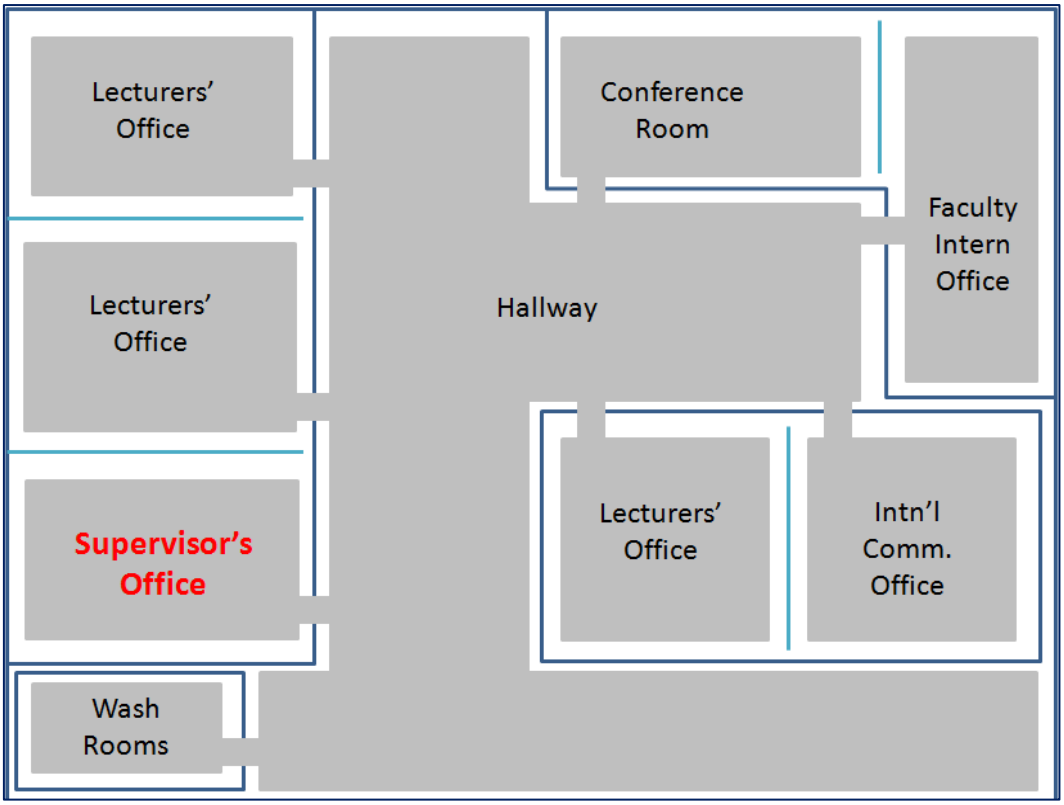

Figure 3.4 – Multiple SLAM Maps with Information Overlay

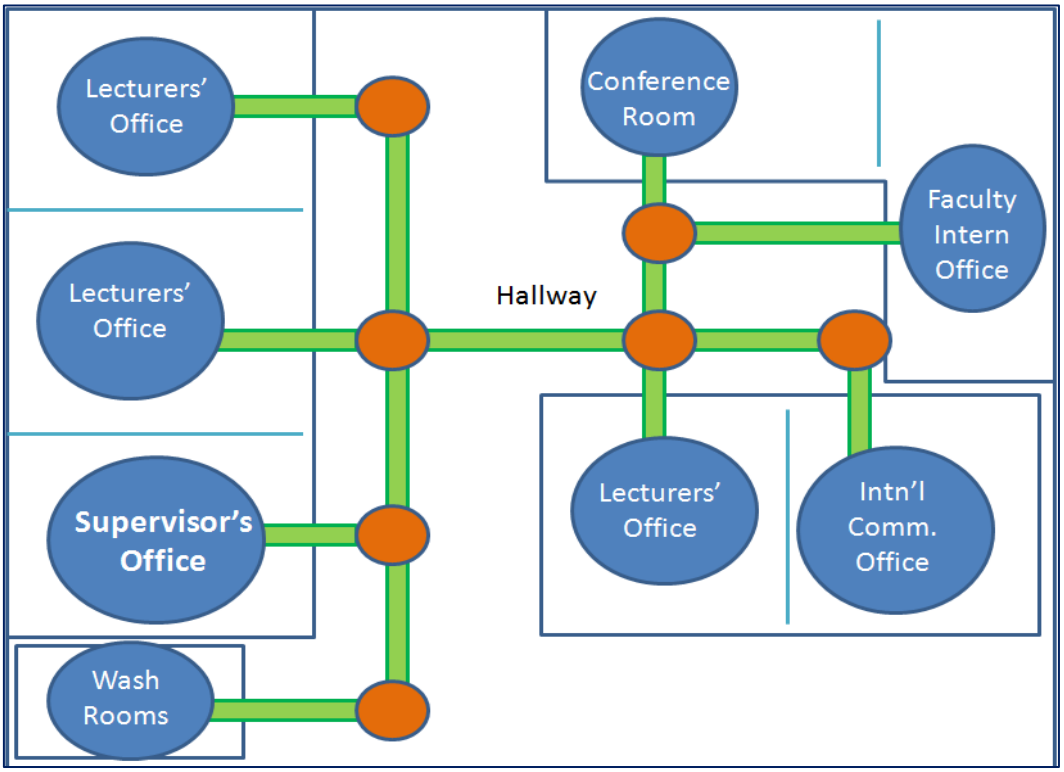

Figure 3.5 – SLAM Map with Graph Abstraction

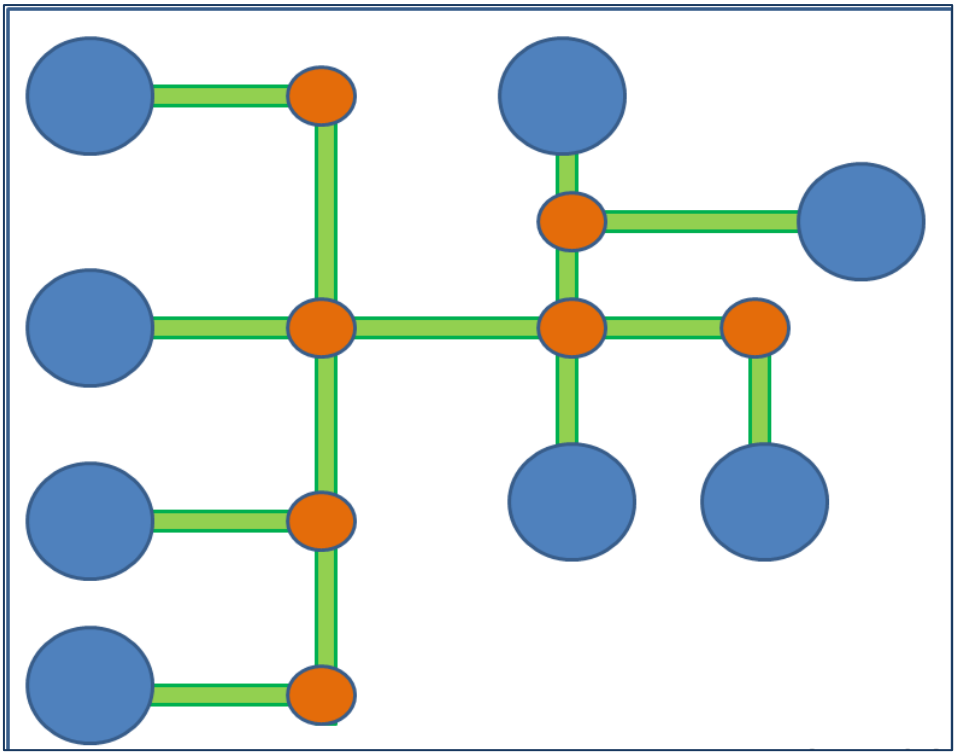

Figure 3.6 - SLAM Maps Graph Model

3.2.2 Localization and Navigation Tools

The Localization and Navigation block is responsible for localization, path planning, and extracting a path from a start point to a destination point for autonomous navigation (refer to Figure 3.7). Destination points are received from service application running on top of this framework with which a user may interact.

- The destination points are simply locations present in the high-level information map of the campus. This navigation command could be for instance instructing the robot to move to the "Faculty Intern Office" assuming its current location is in the "Supervisor's Office". The high-level destination points are therefore, the "Faculty Intern Office" and the "Supervisor's Office".

- The destination location will be interpreted from the natural language form to the corresponding SLAM map coordinates. The interpretation block will determine if the start and destination coordinates belong to different local SLAM maps.
- Having identified that, the Global Path planning routine plans the optimal path between the different nodes of the global map graph. The global path is made up of a series of points that refer to local SLAM maps and coordinate points for the intersections.
- Then for each of the local SLAM maps making up a segment in the global path, a local path is planned.
- The full series of points making up the path from the start to the destination point will be extracted and a movement routine would be started.

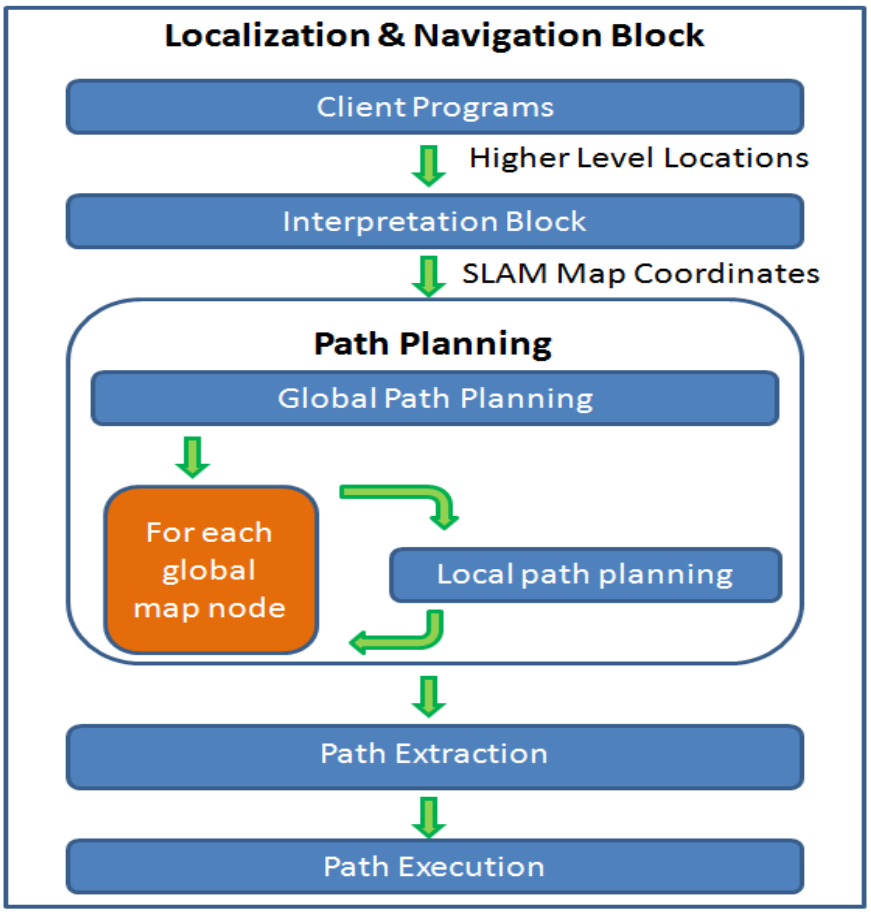

Figure 3.7 – Localization and Navigation Block

3.2.3 Emergency Control Block

During path execution routines to move the robot, the Emergency Control block ensures that the robot does so in a safe manner (refer to Figure 3.8). This block runs obstacle detection and avoidance routines to keep the robot from colliding into objects or moving in a way that would harm the robot.

- The path execution block in this module steps through each of the points from the extracted planned path.
- Before converting them to movement commands, the Emergency Control block runs the various checks to determine if the robot is safe to move.

- The emergency routines firsts check the battery state of the Create base and the laptop placed on the TurtleBot to determine if their power levels are not below a determined threshold valued.
- If the robot passes the power level check, then the distant object detection check is run. This check uses values from the Kinect to identify objects in a defined distant range parameter around the robot.
- If the distant object detection check is passed, then the close object detection check is run. This check uses values from the cliff and bumper sensors to determine if there are unnoticed objects that have gotten very close to the robot. The bumper sensor would determine contact with such object. The cliff sensor will on the other hand determine if the Create base wheels are no longer in contact with the ground. For instance if the robot is put in a place where it could fall from, such as a staircase.
- If any of the emergency checks fails, the stop command module send signals to the actuator drivers to stop the motors of the robot. The stop command module will also send information to alert the user of the inability of the robot to continue moving in a safe manner, until all safety issues are addressed.
- If the emergency checks are all passed, the move command module converts the next path point the robot needs to go to into a movement command for the actuator drivers.
- The actuator drivers send the appropriate velocity and power values to the turn motors via the Create base's microprocessor.

- The robot then localizes itself to know its relation to the objects around and its new position in the map. The ROS implementation of Adaptive Monte Carlo Localization (AMCL) will be used. AMCL is a technique "which uses particle filter to track the pose of the robot against a know map". (Foote, 2013)

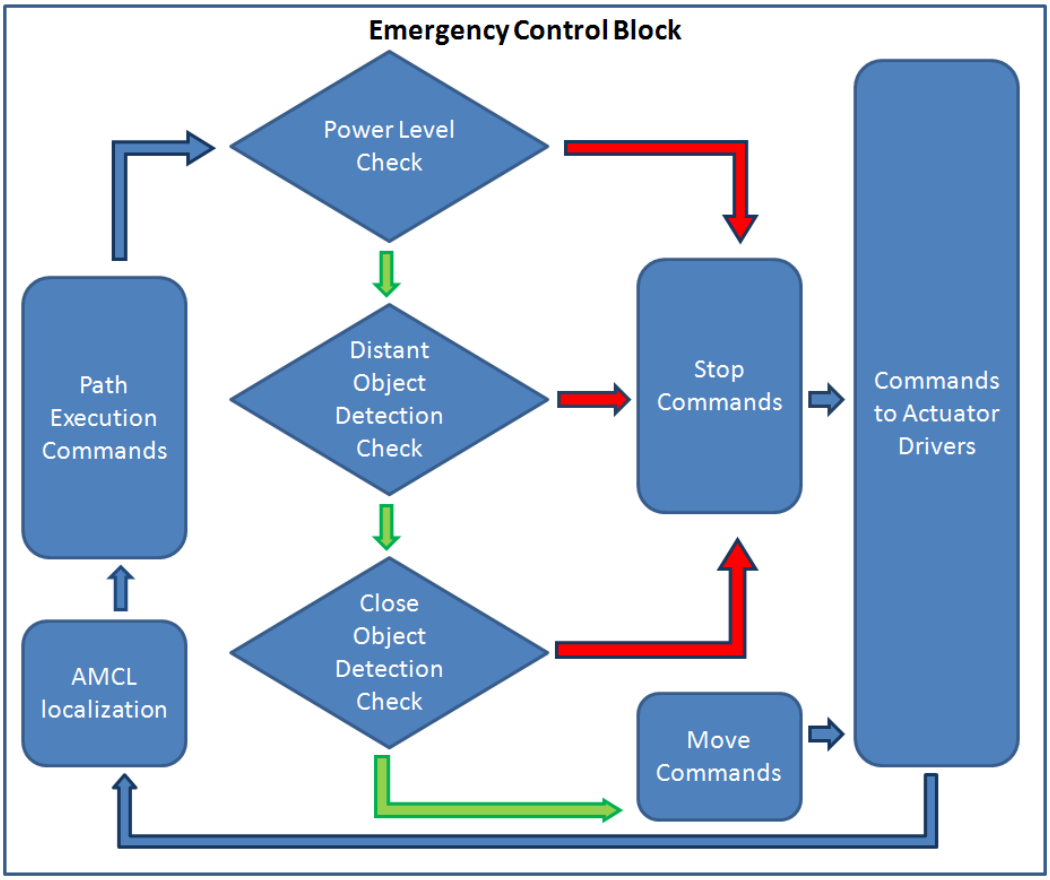

Figure 3.8 – Emergency Control Block

Chapter 4: Implementation and Testing

The precursor to building the proposed context-specific navigation framework is getting the TurtleBot-ROS platform up and running. Most of the time for this project therefore was spent in first understanding the ROS environment and designing the concepts for the system's architecture. The features of the proposed system were not implemented as the TurtleBot was not already set up to a level where it could interact with the ROS platform. Setting up and determining the right packages including the list of dependencies from the ROS platform that were needed for this project was not trivial.

Unfortunately, the tutorial offered by the official ROS.org page was constantly being edited and the page did not have an exhaustive set of instructions to follow in fully installing and configuring ROS. The latest release of ROS; Groovy Galapagos was used for this project. The Groovy installation was completed by using online help, advice from forums and wikis and by simply experimenting and learning in the process.[7]

4.1 Robot Setup

This section covers the procedures used in setting up and configuring the TurtleBot for the project. Appendix A provides a more technical version of the setup process. The final robotic setup was a ROS enabled TurtleBot connected to a workstation computer via an Android WIFI hotspot network as shown in Figure 4.1.

[7] For the official installation tutorial refer to ros.org/wiki/groovy/installation/Ubuntu

Figure 4.1 – TurtleBot Setup over Android WIFI hotspot

4.1.1 Components Used

The final robotic setup was completed using some key components which are listed below.

- TurtleBot robot set (with a Create base, Kinect sensor, Sensor board with 250°/sec gyro sensor).
    - 2 Lenovo T240 ThinkPad laptops (one to run on the TurtleBot and the other for the workstation).
    - An android device to create a WIFI hotspot for the network communication. A WIFI network using an android device was used because the wireless network on campus was not stable enough for constant communication to be maintained between the two laptops on different parts of the campus.
    - 1 8GB USB flash drive to create a bootable Ubuntu version for installation.

4.1.2 Ubuntu 12.04 (Precise)

ROS is a meta-operating system that needs a host Operating System to run on. The latest version of ROS used for the project had its stable

version compatible with Ubuntu only. Thus, Ubuntu had to be downloaded and installed on the two laptops.

- A ubuntu-12.04.2-desktop-i386.iso file was downloaded from the official Ubuntu release page (releases.ubuntu.com/precise/).
- Using a "Make Startup Disk" tool on Linux-Ubuntu virtual machine, a USB bootable flash drive was created from the Ubuntu Precise OS IOS file.
- On each of the laptop computers to be used in the setup, Ubuntu Precise was then installed.
- For easy identification of the machine during network activities, the laptop to be placed on the TurtleBot setup was set to have its hostname (computer name) as 'turtlebot' and the laptop to be used in instructing the TurtleBot has the hostname 'workstation'.
- Both machines have the same password to reduce the overhead cost of constantly remembering and typing passwords for the many administrative oriented commands required on the two laptops.

4.1.3 Installation of ROS Groovy Galapagos

This section describes the steps used in installing the ROS platform

- The first step was to configure Ubuntu to accept software from ROS.org by creating a file in the Ubuntu application sources folder with a reference to *packages.ros.org* software repository.
- Then the keys used in establishing secure communication between the laptop and the ROS.org repository was set up.

- The Ubuntu repository was updated in order to get the newest versions of packages and their dependencies.
- The full desktop Groovy package was downloaded and installed.
- A ROS dependency tool was initialized to allow easy installation of ROS system dependencies.
- The bash environment variables to recognize ROS commands automatically were setup.
- A python based ROS command line tool was installed.
- A network time protocol tool to synchronize network time for the wireless communication was downloaded and installed.
- To allow remote access to the terminal of one computer from a different computer, *openssh server* was installed.
- To establish a ROS Network, two environment variables are very important. The ROS_HOSTNAME that refers to the IP address of a current machine and the ROS_MASTER_URI which refers to the IP address of the machine acting as the master computer in the network setup which in this case is the laptop mounted on the TurtleBot; 'turtlebot'.
- On the android device a WIFI hotspot was setup and the two machines were connected to the WIFI network. The IP addresses of the machines were determined and recorded.
- On the TurtleBot machine ('turtlebot'), the ROS_HOSTNAME and ROS_MASTER_URI were set to point to the IP address of 'turtlebot'. Setting up of the two variables can be very tedious especially when using a network that dynamically assigns IP addresses. Appendix C has a bash script that can be used to simplify the IP setup process.

- On the TurtleBot workstation ('workstation') also, the ROS_HOSTNAME was set to point to the IP address of the 'workstation' and ROS_MASTER_URI to point to the IP address of 'turtlebot'.

4.1.4 Physical Setup

- The TurtleBot kit used for this project was already assembled. However, for assembly instructions refer to TurtleBot manual[8].
- On 'turtlebot' the power management option was set to prevent the computer from automatically going to sleep or hibernating.
- The lid of the computer ('turtlebot') was closed and placed at the bottom shelf of the TurtleBot shelf structure.
- The iRobot Create base was then connected to the 'turtlebot' computer using the mini-DIN connection cable.
- The Kinect power cable was plugged to the sensor board power outlet using the Kinect's MicroFit connection plug.
- And finally the Kinect data USB cable is connected to 'turtlebot' computer on a USB 2.0 port.

[8] TurtleBot assembly instructions can also be found on makezine.com.
http://blog.makezine.com/projects/assemble-a-turtlebot/

4.2 Reconfiguration of ROS System Files

ROS is built in a generic way to support a variety of robot hardware and thus some files have pre-defined values for certain default hardware variables. These values needed to be changed to reflect the particular hardware types that were used in the project. Refer to Appendix B for a more technical document on how to reconfigure ROS system files.

4.2.1 Minimal File

There is a "minimal.launch" file belonging to the "turtlebot_bringup" package for ROS which contains details about all the hardware options that can be connected to a TurtleBot. It has parameters to specify the values for the TurtleBot base, the battery pack, 3D sensor etc. The default values of these parameters were changed to reflect the TurtleBot model being used.

4.2.2 TurtleBot Dashboard File

A "turtlebot_dashboard.launch" file belonging to the ROS visualization package tools; "turtlebot_viz" package was also edited. This file contains details about all the hardware components that are visualized from the "workstation" as the TurtleBot is up and running.

4.2.3 TurtleBot Node File

In the "turtlebot_node.py" file, the values of several parameters were changed to accommodate the stable and tested version of the "turtlebot_create" ROS Groovy packages. The default values that the variables had were from an integrated ROS package for various TurtleBot

bases that had not been fully tested and finally released at the time the system was being set up for this project.

4.2.4 Create Sensor Handler File

The "create_sensor_handler.py" file for "turtlebot_create" ROS Groovy packages has values of parameters that have been set to handle data output from the TurtleBot's sensor pack. Some of these parameters were modified to refer to the appropriate sensor packs available on the TurtleBot kit available for this project.

4.2.5 Python KDL Error Fix

Kinematics and Dynamics Library (KDL) is a library that contains a set of functions that provide an "independent framework for the modeling and computation of kinematic chains for robots, biomechanical human models, computer-animated figures, machine tools, etc." (Kinematics and Dynamics Library). This library makes it easy to specify various kinematics chains, motions and interpolation of geometrical objects (Kinematics and Dynamics Library). Unfortunately, the ROS Groovy that was used in this project came with a bug in the Python edition of the KDL library. The ROS foundation however, provided a separate package as a fix to this issue. This was downloaded separately and installed to fix the bug.

4.3 Testing

After setting up the ROS-TurtleBot system, tests were conducted to ensure that the system was correctly set up and that the ROS packages needed for the framework was functioning correctly. Basically, the tests entailed running the necessary ROS commands to start the ROS packages and assessing their values.

4.3.1 Network Connection

Ubuntu terminal ping tool was used to determine if a connection existed between the two machines over the network. The ROS_HOSTNAME and ROS_MASTER_URI were determined to be pointing to the right IP addresses by echoing them out in the terminal. And also, communication between the ROS environment was tested by using the ROS communication tool ROSTOPIC.

4.3.2 System Recognition

A ROS implemented dashboard tool was used to check if the ROS environment recognized the various hardware and software components in the setup, the dashboard tool was used. As illustrated in the screenshot below (Figure 4.2), the dashboard recorded "OK" for the various devices connected to the ROS platform.

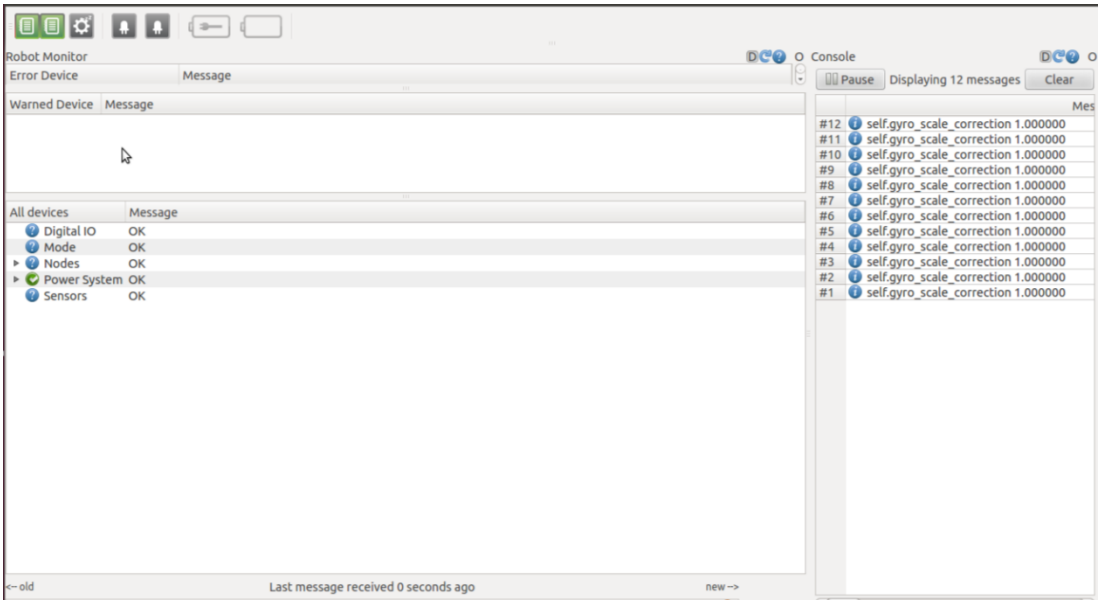

Figure 4.2 – Dashboard View of the TurtleBot System

4.3.3 Vision

For vision, the Kinect kit was activated from ROS to view the data the motion sensor kit was dealing with. In order to verify the depth sensing capabilities of the kit, the TurtleBot was placed in front of a group of working benches and chairs in some part of the computer lab in the university. The first image (Figure 4.3) is a RGB color image of the section lab. Figure 4.4 represents a mono chromatic image of the same view of the lab.

Figure 4.3 – Color Image of Section of Computer Lab

Figure 4.4 – Monochrome Image of Section of Computer Lab

4.3.4 Visualization

The Kinect kit depth sensing capabilities was also tested from the ROS environment. The first image (Figure 4.5) is a depth image of the same section of the lab, where different colors are used to represent objects at different distances from the camera. Color shades from Red to Orange are used to indicate objects in close proximity to the robot. Color ranges from Yellow through Green to Blue and Violet represent objects in order of increasing distance to the robot. On the other hand, the depth image in

Figure 4.6 uses a different variation of gray to represent depth. The darker the shade of gray, the closer an object is while the lighter the shade, the further away an object is.

Using a ROS visualization tool, the robot was viewed in a 3D space where the various depth images were visualized as a Point Cloud image. As shown in Figure 4.7, a model of the robot is placed in a 3D environment with the objects in the robots view represented as a 3D image. In the visualization tool, the 3D images (Point Cloud) can be interacted with in the 3D environment.

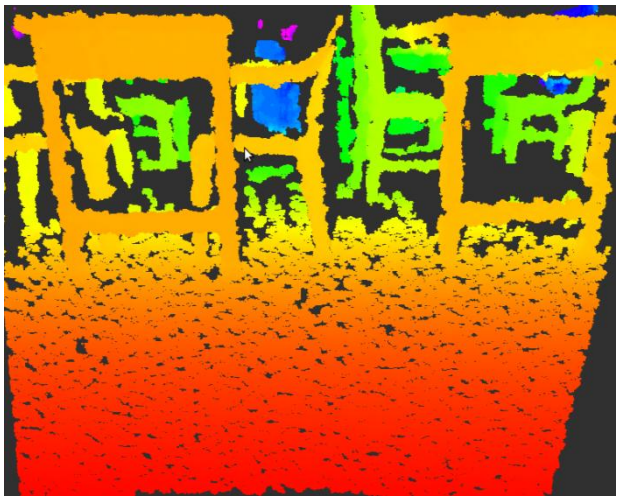

Figure 4.5 – Color Depth Image of Section of Computer Lab

Figure 4.6 - Monochrome Depth Image of Section of Computer Lab

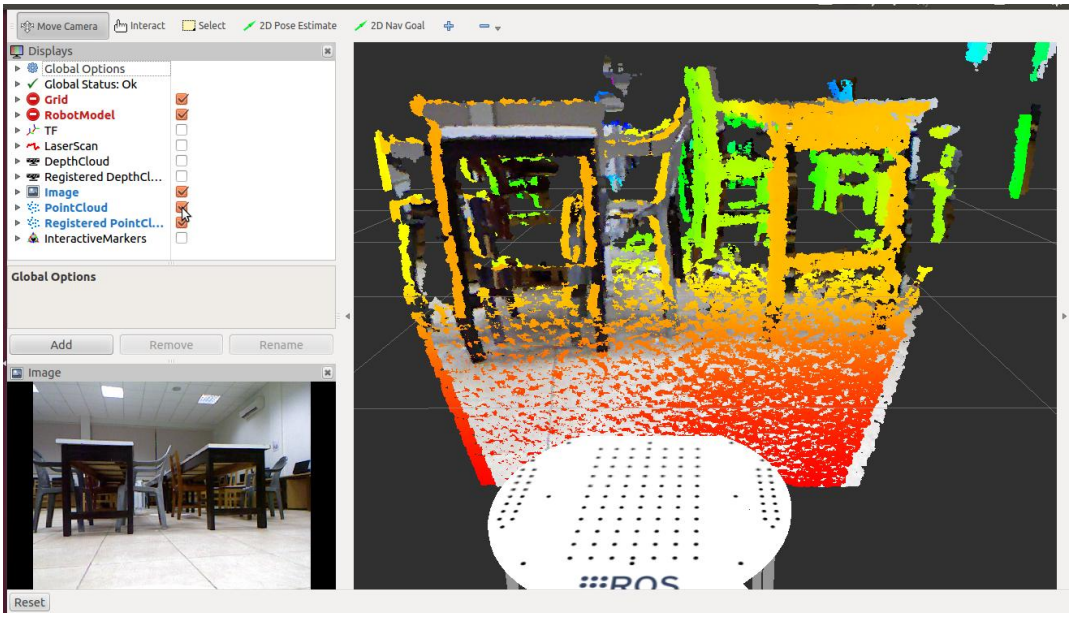

Figure 4.7 – 3D Point Cloud Visualization Image of Section of Computer Lab

4.4 Challenges

Many challenges were encountered in this project but most of these challenges can be summed up into two main categories. These setbacks significantly increased the time used in setting up the robotic system and were quite frankly terribly frustrating.

- Firstly, the project dealt with new technology that had never been used before at Ashesi prior to the project. The TurtleBot kit, the ROS and Ubuntu system were unfamiliar systems that needed a certain level of expertise to use.
- Secondly, there was not sufficient documentation from the ROS website to use as reference. The ROS official page, as mentioned earlier had the information on the page constantly changing. And generally there was not enough information available online to refer to when faced with errors. Many questions also posted on various ROS forums rarely had helpful answers.

Chapter 5: Conclusion and Future Work

This paper described the procedures used in successfully bringing up TurtleBot and ROS to a workable level where the implementation of the customized navigation framework modules can begin. Key ROS dependencies needed for the framework have been tested and found to be functioning properly. Also, concepts of the systems architecture have also being designed, with comprehensive descriptions detailing the functionalities of key modules and relationships existing between them. Detailed instructions on how to set up the TurtleBot ROS system have been documented.

However, more work needs to be done to get this framework to its final state. Work to be done in the future includes;

- Calibrating the gyro sensor for the TurtleBot. Calibration is a very important step to improve the performance of any navigation application by enhancing the quality of feedback given to the system.
- Installing the newly released Java libraries to continue future development in Java which is a relatively simpler environment to work with, as compared to C++, Python, and Bash scripting. For instance, all the main components to be developed for this framework can be done in the Java language.
- Developing the code for the three main components of the proposed navigation framework.

On a concluding note, this project has been a successful one with a great learning curve. Ashesi University College’s goal to experiment with service robotics is in its learning stages and has great prospects for the future.

References

Ackerman, E. (2011,March 07). *Top 10 Robotic Kinect Hacks.* Retrieved March 14, 2013 from IEEE Spectrum Automaton: http://spectrum.ieee.org/automaton/robotics/diy/top-10-robotic-kinect-hacks

Afram, K. A. (2012). Prototyping An Autonomous Mobile Robot Capable of Giving A Tour of Parts of the Ashesi Campus. *Applied Project*, 1-40.

Burgard, W., Cremers, A. B., Fox, D., Hähnel, D., Lakemeyer, G., Schulz, D., . . . Thrun, S. (1999). Experiences With An Interactive Museum Tour-Guide Robot. *Artificial Intelligence*, 3-55.

Dieter Fox, W. B. (1999). Markov Localization for Mobile Robots in Dynamic Environments. *Journal of Artificial Intelligence Research*, 242-246.

Foote, T. (2013, February 22). *ROS.org*. Retrieved January 17, 2013 from http://www.ros.org

Foxy, D., Burgardy, W., & Thrun, S. (1997). The Dynamic Window Approach to Collision Avoidance. *Robotics & Automation Magazine, IEEE*, 23-33.

Fradj, J. (2011, September 29). *Introduction to A* Pathfinding.* Retrieved from raywenderlich.com: http://www.raywenderlich.com/4946/introduction-to-a-pathfinding

Keat, H. W., & Ming, L. S. (2012). An Investigation of The Use of Kinect Sensor For Indoor Navigation. *TENCON 2012 - 2012 IEEE Region 10 Conference*, 1-5.

Kim, G., Chung, W., Kim, K. R., Kim, M., Han, S., & Shinn, R. (2004). The Autonomous Tour-Guide Robot Jinny. *Intelligent Robots and Systems*, 3450-3455.

Kinematics and Dynamics Library. (n.d.). Retrieved March 27, 2013 from Open Robot Control Software Project: http://www.orocos.org/kdl

Quigley, M., Conley, K., Gerkey, B. P., Faust, J., Foote, T., Leibs, J., . . . Ng, A. Y. (2009). ROS: An Open-Source Robot Operating System. *ICRA Workshop on Open Source Software*.

Roy, N., Burgard, W., Fox, D., & Thrun, S. (1998). Coastal Navigation - Robot Motion with Uncertainty. *In AAAI Fall Symposium: Planning with POMDPs* (pp. 135-140). AAAI.

Ruhland, R. (2012, June 19). *Getting Started with ROS (Robotic Operating System).* Retrieved March 09, 2013, from Instructable: http://www.instructables.com/id/Getting-Started-with-ROS-Robotic-Operating-Syste/?ALLSTEPS

Rusu, R. B., Gerkey, B., & Beetz, M. (2008, July 03). Robots In The Kitchen: Exploiting Ubiquitous Sensing And Actuation. *Robotics and Autonomous Systems*, pp. 844–856.

Thrun, S., Bennewitz, M., Burgard, W., Cremers, A., Dellaert, F., Fox, D., . . . Schulz, D. (1999). MINERVA: A Second-Generation Museum Tour-Guide Robot. *Robotics and Automation*, 1999-2005.

Appendix

A. Robot Setup Design

Below are the procedures followed in the setting up, installing and configuring of TurtleBot. ROS version Groovy Galapagos as at 25th of March, 2013 was used to ROS-enable TurtleBot.

A.1 Components Used

- TurtleBot compatible robot set (with a Create base, Kinect sensor, sensor Board with 250°/sec gyro sensor)
- 2 Lenovo T240 ThinkPad laptop computers
- Each laptop with Intel Core i5 processor @ 2.40GHz
- An android device to create a WIFI hotspot
- 1 8GB USB flash drive

A.2 Downloading Ubuntu 12.04 (Precise)

- On the official Ubuntu release page (releases.ubuntu.com/precise/) and at the bottom of the page there is a list of different Ubuntu12.04 release with release options for AMD/Intel processors and 32/64bit systems.
- Click to download ubuntu-12.04.2-desktop-i386.iso file.

A.3 Ubuntu Bootable Flash Drive

- The flash drive is then connected to a Linux-Ubuntu virtual machine running on a Windows OS host.
- The Ubuntu Precise OS is copied to the virtual machine and using the "Make Startup Disk" tool provided by Ubuntu, the Ubuntu

Precise OS IOS file is mounted unto the flash drive and a USB bootable flash drive is created.

- Alternatively, Universal USB Installer for Windows from pendrivelinex.com can be downloaded and used in creating the bootable flash drive.
- The download link is http://www.pendrivelinux.com/downloads/Universal-USB-Installer/Universal-USB-Installer-1.9.0.1.exe

A.4 Installation of Ubuntu Precise

- On each of the laptop computers to be used in the setup, Ubuntu Precise is then installed.
- The laptop to be placed on the TurtleBot setup has the hostname (computer name) as 'turtlebot' and the laptop to be used in instructing the TurtleBot has the hostname 'workstation'.
- Both machines have the same password to reduce the overhead cost of constantly remembering and typing passwords for the many administrative oriented commands required on the two laptops.

A.5 Installation of ROS Groovy Galapagos

Below is the procedure follow to install and configure ROS . For the official installation tutorial refer to ros.org/wiki/groovy/installation/Ubuntu

- First setup Ubuntu to accept software from ROS.org by creating a file in the Ubuntu application sources folder with a reference to packages.ros.org software repository. This can be done with the command below in the Ubuntu terminal.

```
o  sudo sh –c ‘echo “deb http://packages.ros.org/ros/ubuntu
   precise main” > /etc/apt/sources.list.d/ros-latest.list’
```

- Set up the keys used in establishing secure communication between the laptop and the ROS.org repository.

```
o  wget http://packages.ros.org/ros.key -O - | sudo apt-key add
   –
```

- Update the Ubuntu repository and update to get the newest versions of packages and their dependencies.

```
o  sudo apt-get update
```

- Download and install the full desktop Groovy package.

```
o  sudo apt-get install ros-groovy-desktop-full
```

- Initialize a tool to allow easy installation of ROS system dependencies.

```
o  sudo rosdep init
o  rosdep update
```

- Setup the bash environment variables to recognize ROS commands automatically.

```
o  echo “source /opt/ros/groovy/setup.bash” >> ~/.bashrc
o  source ~/.bashrc
```

- Install a python based ROS command line tool.

```
o  sudo apt-get install python-rosinstall
```

- Install network time protocol tool to synchronize network time for the wireless communication.

```
o  sudo apt-get install chrony
o  sudo ntpdate ntp.ubuntu.com
```

- Install openssh server to allow remote access to the terminal of one computer from a different computer.

```
o  sudo apt-get install openssh-server
```

- To establish a ROS Network, two environment variables are very important; the ROS_HOSTNAME that refers to the IP address of a current machine and the ROS_MASTER_URI which refers to the IP address of the machine acting as the master computer. In this network setup the laptop mounted on the TurtleBot; 'turtlebot' is the master.
- On the android device set up WIFI hotspot and connect the two machines to the network.
- The IP addresses of the two machines need to be determined and recorded.

```
o  ifconfig
```

- On the TurtleBot machine 'turtlebot', set up the ROS_HOSTNAME and ROS_MASTER_URI to point to the IP address of the computer.

```
o  sed -i '/ROS_MASTER_URI=/ d' ~/.bashrc
o  sed -i '/ROS_HOSTNAME=/ d' ~/.bashrc
o  echo export ROS_MASTER_URI=http://IP_OF_TURTLEBOT:11311 >>
   ~/.bashrc
o  echo export ROS_HOSTNAME=IP_OF_TURTLEBOT >> ~/.bashrc
o  source ~/.bashrc
```

- On the 'workstation' machine, setup ROS_HOSTNAME to point to the IP address of the computer and ROS_MASTER_URI to point to the IP address of 'turtlebot'.

```
o  sed -i '/ROS_MASTER_URI=/ d' ~/.bashrc
```

```
o  sed -i '/ROS_HOSTNAME=/ d' ~/.bashrc
o  echo export ROS_MASTER_URI=http://IP_OF_TURTLEBOT:11311 >>
   ~/.bashrc
o  echo export ROS_HOSTNAME=IP_OF_WORKSTATION >> ~/.bashrc
o  sed -i '/ROS_MASTER_URI=/ d'/opt/ros/groovy/setup.sh
o  sudo sed -i '/ROS_HOSTNAME=/ d'/opt/ros/groovy/ setup.sh
o  sudo echo "export
   ROS_MASTER_URI=http://IP_OF_WORKSTATION:11311" >>
   /opt/ros/groovy/setup.sh
o  sudo echo "export ROS_HOSTNAME=IP_OF_TURTLEBOT" >>
   /opt/ros/groovy/setup.sh
o  source ~/.bashrc
```

- Refer to Appendix C for simple bash script that can be used to simplify that assignment of values to the ROS_MASTER_URI and ROS_HOSTNAME variables.
- Verify that the values of the environment variables of the two machines have the correct IP addresses assigned to them.

```
o  echo $ROS_HOSTNAME
o  echo $ROS__MASTER_URI
```

A.6 Physical Setup

- Assemble the TurtleBot kit if it isn't already assembled. Refer to TurtleBot manual for assembly instructions[9].
- On 'turtlebot' set the power management option to prevent the computer from automatically going to sleep or hibernating.

[9] TurtleBot assembly instructions can also be found on makezine.com. http://blog.makezine.com/projects/assemble-a-turtlebot/

- Close the lid of the computer and place it at the bottom shelve of the TurtleBot robot stack of shelves.
- Connect the iRobot create base to the 'turtlebot' computer using the mini-DIN connection cable.
- Plug the Kinect power cable to the sensor board power MicroFit connection.
- Plug the Kinect data USB cable to 'turtlebot' computer on a USB 2.0 port.

B. Reconfiguration of ROS System Files

ROS is built in a generic way to support a variety of robot hardware and thus some files have pre-defined values for certain default hardware variables. These values were changed to reflect the particular hardware types that are was used in the project.

B.1 Minimal File

There is a "minimal.launch" file belonging to the "turtlebot_bringup" package for ROS in this path: "/opt/ros/groovy/stacks/turtlebot/turtlebot _bringup/launch". This file contains details about all the hardware options that can be connected to a TurtleBot. It has parameters to specify the values for the TurtleBot base, the battery pack, 3D sensor etc.

- If the "turtlebot_bringup" package directory does not exist, download from packages.ros.org.

```
o  sudo apt-get install ros-groovy-turtlebot-bringup
```

- Using administrative privileges the "minimal.launch" file on the "turtlebot" machine can be opened in the gedit text editor.

```
o  sudo gedit /opt/ros/groovy/stacks/turtle/turtlebot_
   bringup/launch/minimal.launch
```

- For this project the value of the following arguments in the minimal.launch file need to be changed to the values below.

```
o  <arg name="base"    value="$(optenv TURTLEBOT_BASE
   create)"/>
o  <arg name="battery"   value="$(optenv TURTLEBOT_
   BATTERY /proc/acpi/battery/BAT0)"/>
o  <arg name="stacks"    value="$(optenv TURTLEBOT_
   STACKS circles)"/>
o  <arg name="3d_sensor" value="$(optenv TURTLEBOT_
   3D_SENSOR kinect)"/>
o  <arg name="simulation" value="$(optenv TURTLEBOT_
   SIMULATION false)"/>
```

B.2 TurtleBot Dashboard File

There is a "turtlebot_dashboard.launch" belonging to the TurtleBot visualization package tools; "turtlebot_viz" package in this path: "/opt/ros/groovy/stacks/turtlebot_viz/turtlebot_dashboard_launchers/ launch". This file contains details about all the hardware components that are visualized from the "workstation" as the TurtleBot is up and running.

- If the "turtlebot_viz" package directory does not exist, download from packages.ros.org.

```
o  sudo apt-get install ros-groovy-turtlebot-viz
```

- Using administrative privileges the "turtlebot_dashboard.launch" files on both the "turtlebot" and "workstation" machine can be opened in the gedit text editor.

```
o  sudo gedit /opt/ros/groovy/stacks/turtlebot_viz/turt
   lebot_dashboard_launchers/turtlebot_dashboard.launch
```

- Then the TurtleBot base argument value is changed to "create".

```
o  <arg name="base"    value="$(optenv TURTLEBOT_BASE
   create)"/>
```

B.3 TurtleBot Node File

In the "turtlebot_node.py" file under "/opt/ros/groovy/turtlebot_create/ create_node/nodes/" directory, the values of several parameter should changed to accommodate the stable and tested version of the "turtlebot_create" ROS Groovy packages. The default values that the variables had, were from an integrated ROS package for various TurtleBot bases that had not been fully tested and finally released at the time of the project.

- If the "turtlebot_create" package directory does not exist, download from packages.ros.org.

```
o  sudo apt-get install ros-groovy-turtlebot-create
```

- Using administrative privileges the "turtlebot_dashboard.launch" files on both the "turtlebot" and "workstation" machine can opened in the gedit text editor.

```
o  sudo gedit /opt/ros/groovy/stacks/turtlebot_create
   /create_node/nodes/turtlebot_node.py
```

- Line 56 of the file should be changed by replacing "roslib.rosenv" with "rospkg" so that it reads the following below.

```
o  import rospkg
```

- Line 65 of the file should also be changed by replacing "turtlebot_driver" with "create_driver" so that it reads the following below.

```
o  from create_driver import TurtleBot,MAX_WHEEL_SPEED,
   DriverError
```

- Line 509 of the file is also changed by replacing "roslib.rosenv" to "rospkg" so that it reads the following below.

```
o  def connected_file(): return os.path.join(rospkg.get
   _ros_home(), 'turtlebot-connected')
```

B.4 Create Sensor Handler File

The "create_sensor_handler.py" file under "/opt/ros/groovy/turtlebot_create/create_node/src/create_node/" directory has values of several parameter that have been set to handle data output from the TurtleBot's sensor pack.

- If the "turtlebot_create" package directory does not exist, download from packages.ros.org

```
o  sudo apt-get install ros-groovy-turtlebot-create
```

- Using administrative privileges the "turtlebot_dashboard.launch" files on both the "turtlebot" and "workstation" machine can be opened in the gedit text editor.

```
o  sudo gedit /opt/ros/groovy/stacks/turtlebot_create/
   create_node/src/create_node/create_sensor_handler.py
```

- Line 40 of the file is changed by replacing "turtlebot_driver" with "create_driver" so that it reads the following below.

```
o  from create_driver import SENSOR_GROUP_PACKET _LENGTHS
```

B.5 Python KDL Error Fix

Kinematics and Dynamics Library (KDL) contains a set of functions that provide an "independent framework for the modeling and computation of kinematic chains, for robots, biomechanical human models, computer-animated figures, machine tools, etc" (Kinematics and Dynamics Library). This library makes it easy to specify various kinematics chains, motions and interpolation of geometrical objects (Kinematics and Dynamics Library). Unfortunately, the ROS Groovy that was used in this project came with a bug in the Python edition of the KDL library. The ROS foundation however, provided a separate package as a fix to this issue.

- The Python version of the KDL library was downloaded from packages.ros.org

```
o  sudo apt-get install ros-groovy-python-orocos-kdl
```

C. Network Bash Script

This section has the code for a simply script that could be run to simply the procedure for changing the IP address values for ROS_MASTER_URI and ROS_HOSTNAME of two the laptops used in this project with the hostname 'turtle' and 'workstation'. The code can be copied and saved into a text file with the extension ".sh". The file should then be converted to an executable using the CHMOD terminal command before using.

```
#!/bin/bash

hosttype=`hostname`

echo "Hostname of this machine is $hosttype"

if [ $hosttype = turtlebot ]

     then

     read -p "Enter the ip-address of the turtlebot CLIENT
machine : " master_uri

     sed -i '/ROS_MASTER_URI=/ d' ~/.bashrc

     echo "export ROS_MASTER_URI=http://$master_uri:11311" >>
~/.bashrc

     sed -i '/ROS_HOSTNAME=/ d' ~/.bashrc

     echo "export ROS_HOSTNAME=$master_uri" >> ~/.bashrc

     else

     read -p "Enter the ip-address of the turtlebot CLIENT
machine : " master_uri

     sed -i '/ROS_MASTER_URI=/ d' ~/.bashrc

     echo "export ROS_MASTER_URI=http://$master_uri:11311" >>
~/.bashrc

     read -p "Enter the ipaddress of the turtlebot WORKSTATION
machine : " theHostname

     sed -i '/ROS_HOSTNAME=/ d' ~/.bashrc
```

```
      echo "export ROS_HOSTNAME=$theHostname" >> ~/.bashrc

      sudo sed -i '/ROS_MASTER_URI=/ d'/opt/ros/groovy/setup.sh

      sudo sh -c 'echo "export ROS_MASTER_URI=http://'$master
_uri:11311'" >> /opt/ros/groovy/setup.sh'

      sudo sed -i '/ROS_HOSTNAME=/ d' /opt/ros/groovy/setup.sh

      sudo sh -c 'echo "export ROS_HOSTNAME='$theHostname'" >>
/opt/ros/groovy/setup.sh'

fi

echo "Done"
```